

## **7067CEM Coursework Submission**

Group 6:

1. NOMAAN SAYED (SID: 13150622)
2. KINGSLEY OGBUKWU (SID: 13979160)
3. OMAR ALZUBAIDI (SID: 10592454)

**Module Leader: Dr. Amjad Saeed Khan**

**August 2023.**

## Contents

Introduction .....	3
System Design .....	5
Refresh frequency for the seven-segment display: .....	5
Reaction state machine .....	5
Implementation .....	6
VHDL Code Development: .....	6
Circuit Synthesis with diagram: .....	6
Calculating the clock divider: .....	6
Power consumption analysis: .....	7
Comparator:- .....	8
Comparator (four bit latch):- .....	9
Counter: .....	10
Random generator: .....	10
Milsecond counter:- .....	11
Simulation .....	12
Testbench Development: .....	12
Comparator testbench .....	12
Seven segment testbench .....	13
Mileseconter testbench .....	14
Random generator testbench .....	15
Clock divider testbench .....	16
Design Verification: .....	17
conclusion .....	18
appendix .....	19

## Introduction

The project demonstrates the knowledge and understanding of hardware design and VHDL programming skills. The design is a timer circuit which measures how fast human hand can respond after a person sees a visual stimulus. The circuit has three input push buttons corresponding to the start, clear, stop. It uses a single discrete LED as the visual stimulus and displays relevant information on the seven-segment LED display. The user pushes the clear button which forces the circuit to return to the initial state, in which the seven-segment display will show a welcome message as 'HI', and the stimulus LED is off. Then user pushes the start button to initiate the test. The seven segments will go off. After a random interval of 2 to 15 seconds the stimulus goes on and the timer starts to count upward. The timer increases every millisecond, and its value is displayed in the format of "0000" millisecond on the seven-segment LED. After the stimulus LED goes on, the user should try to push the stop button as soon as possible. The timer pauses counting once the stop button is asserted.

The seven-segment LED shows the response time. It should be around 150 to 300 milliseconds for most people. If the stop button is not pushed, the timer stops after 1 second and displays "1000". 7. If the stop button is pushed before the stimulus LED goes on, the circuit displays "9999" on the seven-segment LED and stops. The assumed clock is 100 mhz. The report consists of diagram which explains the functionality and performance of the circuit. It also gives the depth analysis of the power consumption and step-by-step explanation of the code.

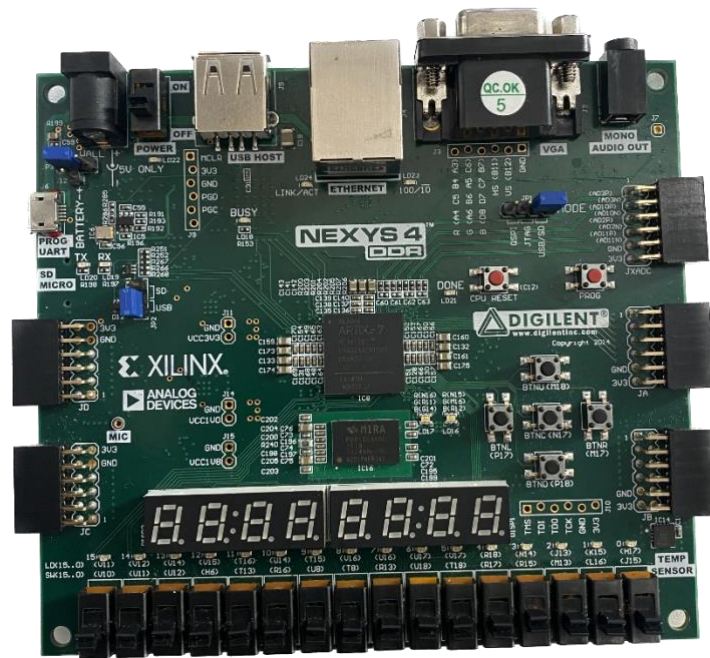


Figure 1. NEXYS 4 DDR

## System Design

### Refresh frequency for the seven-segment display:

The seven-segment display was multiplexed to display a different digit since it is connected via a shared anode. This indicates that the anode pins must be switched on/off quickly enough to give the impression that all the digits are on.

In several systems, state machines are often utilised. The response module uses a state machine, which accepts inputs to modify states and outputs, to control every module in the system.

### Reaction state machine

Five states have been added to the state machine according to the original specification.

Idle: Waits for the start button to be hit while showing "HI" on the seven-segment display and turning off the LED.

Start: Turns off the seven-segment display, reads the random number, and sends it to the random interval module. When the interval is up, this will wait in seconds and deliver a signal to the start state. The millisecond counter will begin, and the user LED will illuminate. The status will change to invalid if the stop button is hit before the returned signal.

React: Activating the react state will activate the seven-segment display, displaying the incrementing counter. When the clear button is pushed, the status is reset to idle. If you press the stop button, the status changes to the stop state.

Stop: When the stop button is pressed, the counter stops counting, and the time is displayed on the seven-segment display. Pressing the clear button returns the status to idle.

Invalid: When in the invalid state, the seven-segment display displays "9999" and returns to idle when the clear button is pressed.

## Implementation

### VHDL Code Development:

The signals in the entity that will link to the outside world as well as all the preceding modules are exchanged between each other via a top module that connects them all. Each module must be instantiated, along with its entity declaration and mapping of any generics or port mappings. This top module provides the generic values that some of the other modules employ, which enables the many instantiations of the same module to perform differently.

Using Vivado, it is possible to estimate how much energy and resources the design consumes. There are two types of power: static and dynamic. Since the system is not in use, the static power is substantially lower than planned. When the system is operating, the power output is 1.205W, or 93% of the total. The I/O powers consume 1% of the dynamic power because there isn't much contact with the I/O. The modules' utilisation of internal logic and internal signals accounts for half of the dynamic power that is still available.

### Circuit Synthesis with diagram:

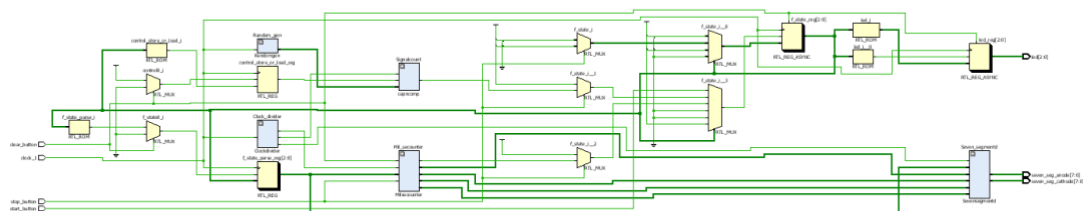


Figure 2. Complete elaborated diagram of the system

### Calculating the clock divider:

Implementing a counter that, whenever it reaches a certain value, changes the status of a std\_logic signal (which, once the count value is achieved, will either be zero or

one), is the simplest approach to create a clock. This's implementation is covered in the chapter on implementation. Equation 1 illustrates the formula for determining what value to count to. There are three or four separate clocks in this setup. The Nexys-4 DDR operates on a 100MHz clock, which is followed by a 1ms (1000Hz) clock used to increase the millisecond counter, a refresh frequency for the seven-segment display (described below), and finally a 1 second counter used for the random interval to start the programme. The circuit is made up of number of flip-flops, counters, logic gates. The purpose of a clock divider circuit is take an input clock signal and produces an output clock signal which is a fraction of an input clock frequency. This can be useful in generating lower frequency clock signal for power saving purposes.

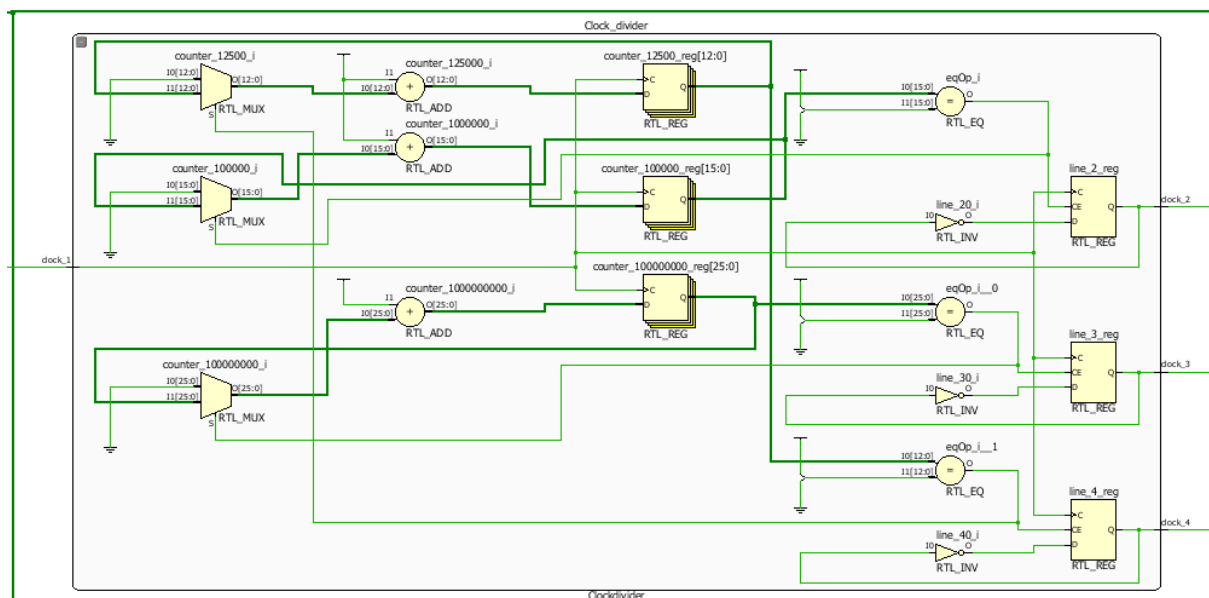


Figure 3. Clock divider

### Power consumption analysis:

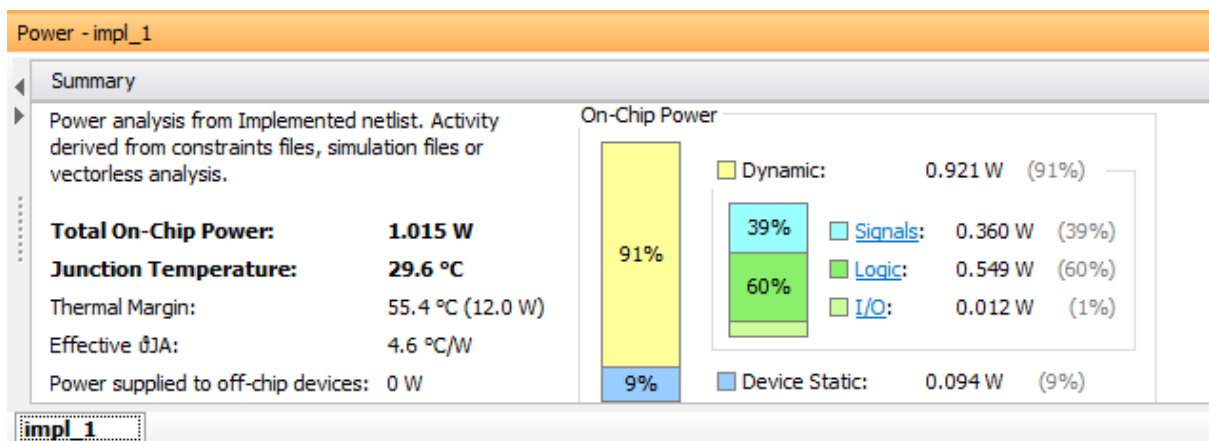


Figure 4. Power Consumption Analysis

The figure above shows that design has a total on-chip power of 1.015 W, which is the sum of the dynamic and static power consumption of the design. The dynamic power is the power consumed by the switching activity of the design, while the static power is the power consumed by the leakage current of the design. The table also shows that the design has a dynamic power of 91% and a static power of 9%.

The table also shows that the design has a junction temperature of 59.6 °C, which is the temperature at the hottest point of the FPGA. The junction temperature depends on the ambient temperature, the airflow, the heat sink, and the power dissipation of the design. The table also shows that the design has a thermal margin of 29.6 °C, which is the difference between the junction temperature and the maximum operating temperature of the FPGA. The higher the thermal margin, the better the reliability and performance of the design.

The table also shows that the design has a power supplied to off-chip devices of 0 W, which is the power consumed by the external components connected to the FPGA, such as memory, sensors, or LEDs. The table also shows that the design has an effective tJA of 45.6 °C/W, which is the thermal resistance between the junction and the ambient air. The lower the tJA, the better the heat dissipation of the design.

#### Comparator:-

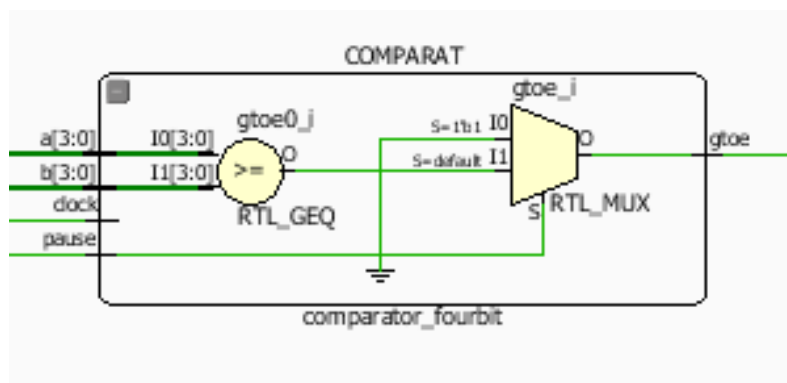


Figure 5. Four bit comparator

gtoe – greater than or equal to



The comparator compares two voltages or currents and outputs a digital signal indicating which is larger. In this circuit, it will compare two random inputs and give the output of the largest number out of two. The multiplexer selects multiple input and select one input which is forwarded to the output. In case of comparator, multiplexer is used to select which input is compared by the comparator, this can be useful in condition when multiple comparisons need to be made and only one comparator is available. By using multiplexer it reduces the number of comparator needed which reduces its complexity. The output from RTL\_GEQ becomes the input of the multiplexer, it can be more than one. It gives the output at gtoe by finding greater than or equal to the input.

#### Comparator (four bit latch):-

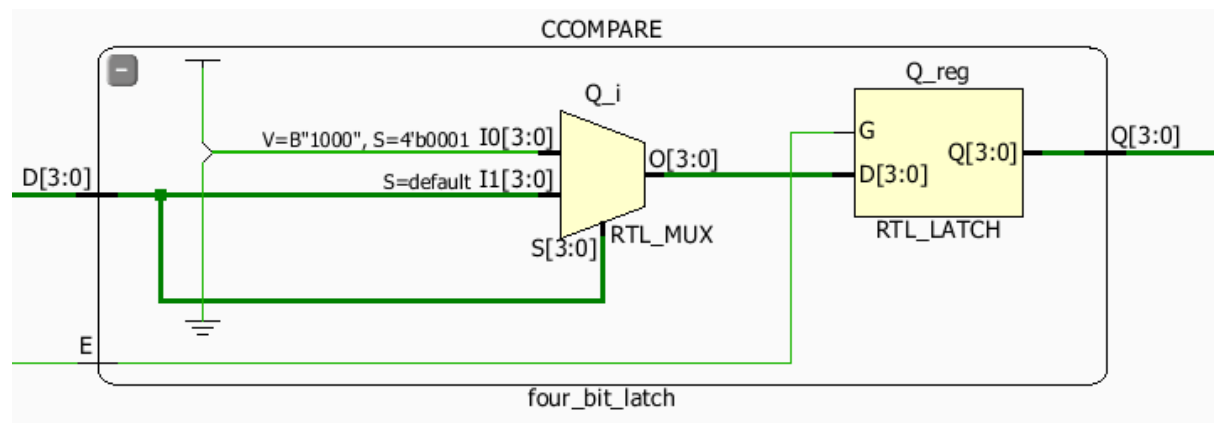


Figure 6. Four bit latch

The latch is a device that uses feedback loops to store one bit of information (either 0 or 1). A latch has two stable states that can be switched by applying signals to its inputs. As long as the latch is powered on, it can retain its state even after the input signals are removed. The four bit latch receives four inputs D into the multiplexer, which then compares the various input scenarios. There is also a feedback loop, which sends direct feedback to the input and the enable, which in turn enables the latch for processing multiple output.

## Counter:

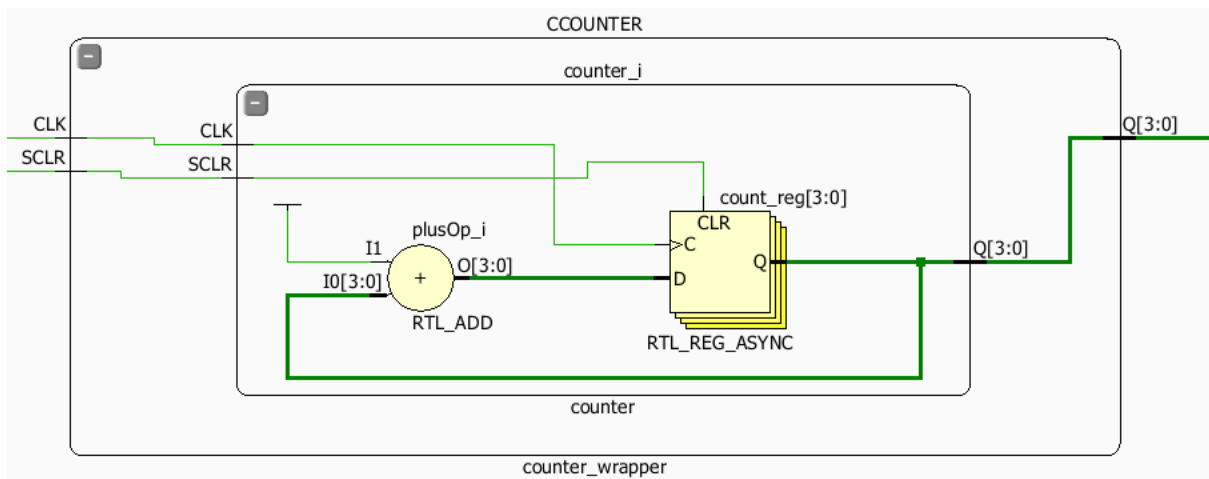


Figure 7. Counter

This is asynchronous counter which depends on the input clock, the count wrapper flips the number while the counter I sustains the required digit, the adder is used to increase the number ( $n + 1$ ).

## Random generator:

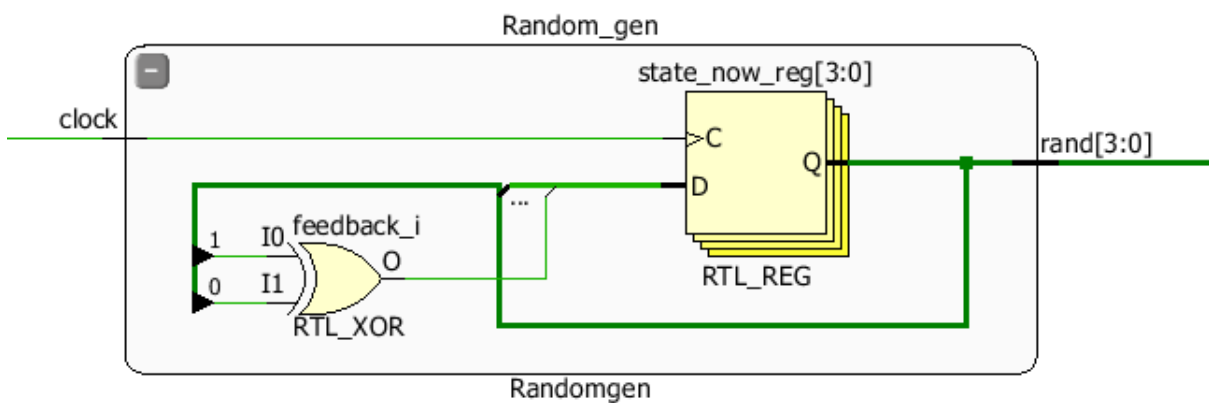


Figure 8. Random Generator

The circuit works by using a feedback loop to generate random numbers. The circuit starts with a **state\_now\_reg[3:0]** input, which is a 4-bit binary number that represents the current state of the circuit. The **state\_now\_reg[3:0]** input is fed into a register labeled "RTL REG" with a "C" and "D" input and a "Q" output. The "C" input of the register is connected to a **clock** labeled "clock". The clock generates a pulse that causes the register to store the value of the **state\_now\_reg[3:0]** input. The output of the register is then fed into an XOR gate labeled "RTL XOR" with two

inputs and one output. One of the inputs of the XOR gate is connected to the output of the register, while the other input is connected to a feedback loop labeled “feedback\_i”. The output of the XOR gate is then fed back into the feedback loop, creating a cycle. The output of the XOR gate is also connected to a rand[3:0] output

Millisecond counter: -

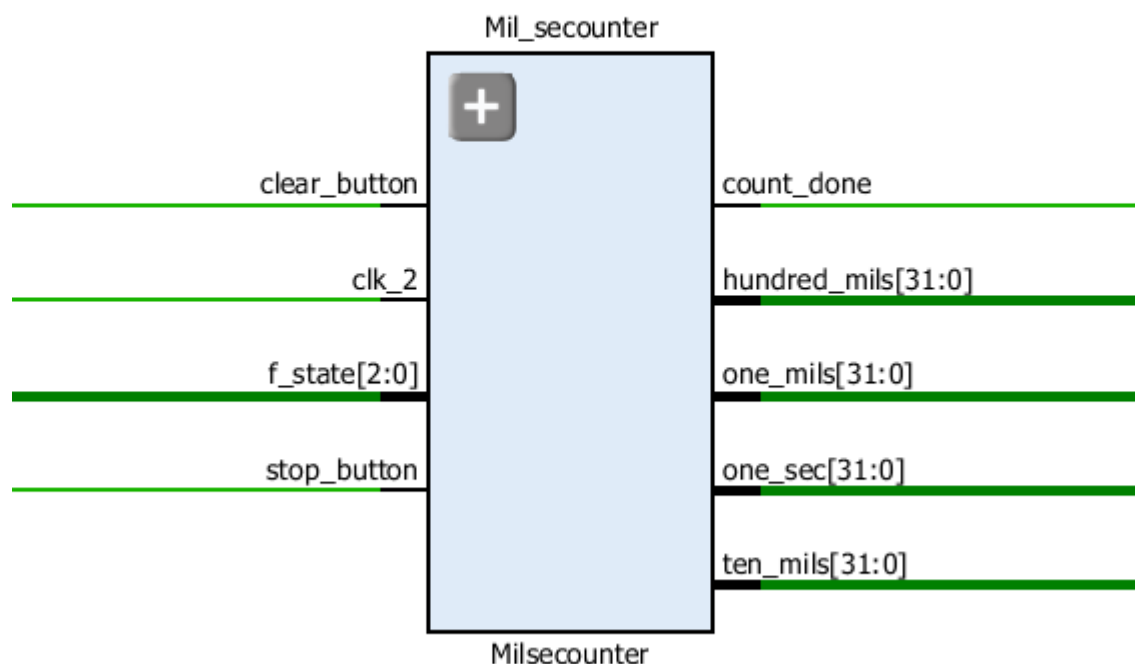


Figure 9. Millisecond counter

The millisecond counter uses the clear button to clear the default HI screen, which now activates the clock; and then it moves to the next state and the LED goes on and starts counting in upward direction with the stimulus LED as ON state, the counting can be in 100ms, 10ms and that depends how fast a human can press the stop button, after pressing the stop button it'll move back to the initial state.

## Simulation

A crucial idea to keep in mind while creating code for FPGA is that VHDL (or other languages) is a hardware description language, not a programming language. Testbenches are used to test the hardware, like how unit tests for high level languages are used. The output signals from the provided input signals are tested here. Every module should have a testbench attached to it so that users may simulate how the hardware will function after being loaded into the FPGA. The complete testbench programmes for each module are included in the Appendix, and the following sections provide the simulation results.

### Testbench Development:

#### Comparator testbench

```
1
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7 entity comparator_fourbit is
8   port (
9       a:      in std_logic_vector(3 downto 0);
10      b:      in std_logic_vector(3 downto 0);
11      gtoe:   out std_logic;
12      clock:  in std_logic;
13      pause:  in std_logic
14   );
15 end comparator_fourbit;
16
17 architecture Behavioral of comparator_fourbit is
18
19 begin
20
21   process(clock, pause, a, b)
22   begin
23       if (pause='1') then
24           gtoe <= '0';
25       elsif (a >= b) then
26           gtoe <= '1';
27       else
28           gtoe <= '0';
29       end if;
30
31   end process;
32
33 end Behavioral;
```

## Seven segment testbench

```
10 entity Sevenssegmentd is
11   Port (clk_4 : in std_logic;
12         one_mils : in integer;
13         ten_mils : in integer;
14         hundred_mils : in integer;
15         one_sec : in integer;
16         f_state : in std_logic_vector(2 downto 0);
17         anode : out std_logic_vector(7 downto 0);
18         cathode : out std_logic_vector(7 downto 0)
19   );
20 end Sevenssegmentd;
21
22 architecture Behavioral of Sevenssegmentd is
23 begin
24   process(clk_4)
25     variable seg_sel: unsigned (1 downto 0) := "00";
26   begin
27     if rising_edge(clk_4) then
28       if f_state = "000" then
29         case (seg_sel) is
30           when "00" =>
31             anode <= "11110111";
32             cathode <= "11111111";
33           when "01" =>
34             anode <= "11111101";
35             cathode <= "10010001";
36           when "10" =>
37             anode <= "11111110";
38             cathode <= "11110011";
39           when "11" =>
40             anode <= "11111110";
41             cathode <= "11111111";
42           when others =>
43             anode <= "11110000";
44             cathode <= "11111110";
45         end case;
46         seg_sel := seg_sel + 1;
47       elsif f_state = "001" then
48         anode <= "11110000";
49         cathode <= "11111111";
50       elsif ((f_state = "010") or (f_state = "100")) then
51         case (seg_sel) is
52           when "00" =>
53             anode <= "11110111";
54           case (one_sec) is
55             when 0 =>
56               cathode <= "00000011";
```

Name	Value		800 ns	900 ns	1,000 ns	1,100 ns	1,200 ns
clk_4	1						
one_mils	1			-2147483648		1	
ten_mils	1			-2147483648		1	
hundred_mils	-2147483648				-2147483648		
one_sec	-2147483648				-2147483648		
f_state[2:0]	U			U			
anode[7:0]	UU			UU			
cathode[7:0]	UU			UU			

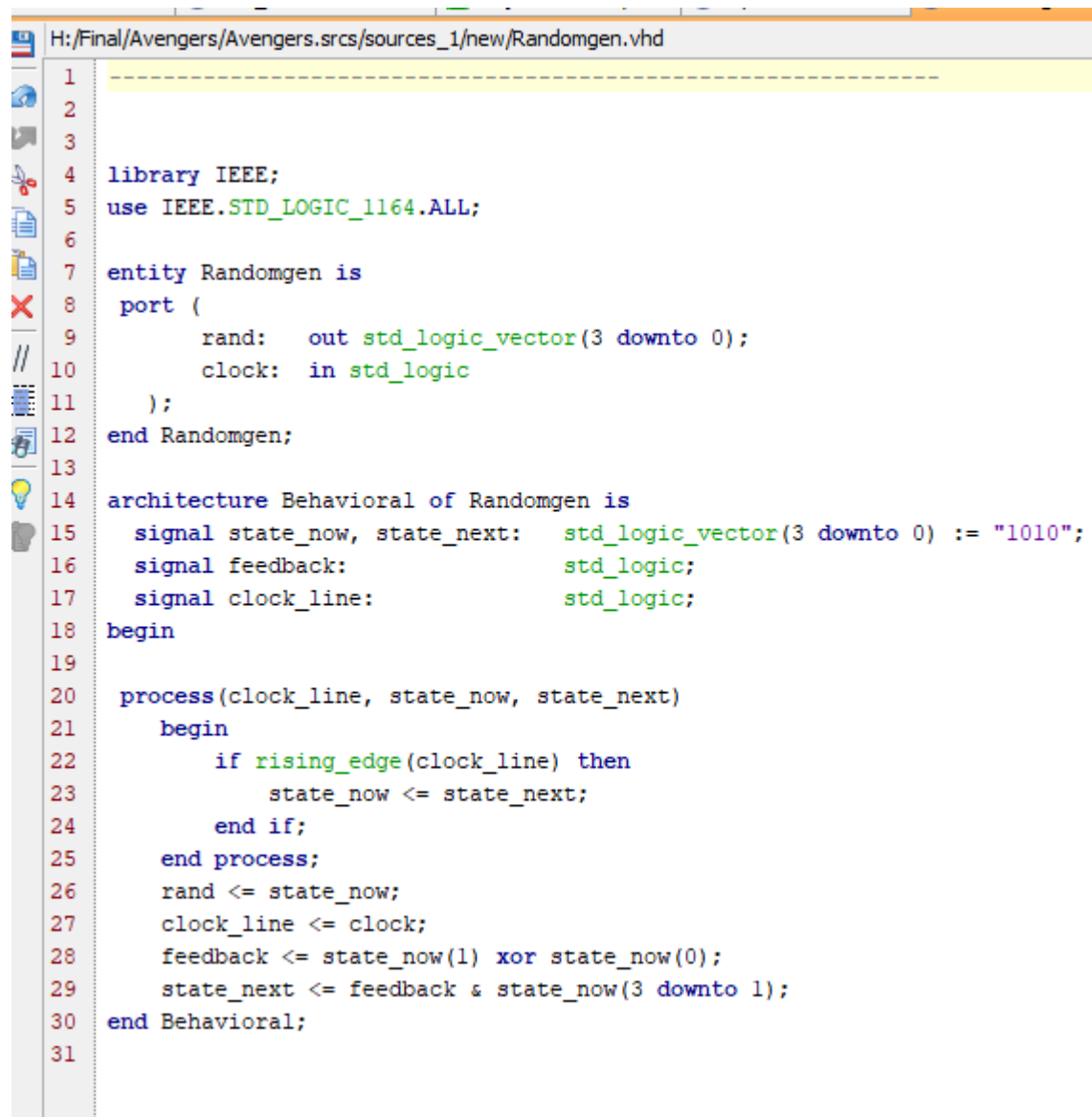
## Mileseconter testbench

```

H:/Final/Avengers/Avengers.srcs/sources_1/new/Milsecounter.vhd
9  Port (clk_2 :in std_logic;
10      clear_button : in std_logic;
11      stop_button : in std_logic;
12      f_state : in std_logic_vector(2 downto 0);
13      one_mils : out integer;
14      ten_mils : out integer;
15      hundred_mils : out integer;
16      one_sec : out integer;
17      count_done : out std_logic
18  );
19  end Milsecounter;
20
21  architecture Behavioral of Milsecounter is
22  begin
23  Milsecounter: process (clk_2, clear_button, stop_button, f_state)
24  variable one_millisecond : integer := 0;
25  variable ten_millisecond : integer := 0;
26  variable hundred_millisecond : integer := 0;
27  variable thousand_millisecond : integer := 0;
28  begin
29      if (clear_button='1') then
30          one_millisecond := 0;
31          ten_millisecond := 0;
32          hundred_millisecond := 0;
33          thousand_millisecond := 0;
34          one_mils <= 0;
35          ten_mils <= 0;
36          hundred_mils <= 0;
37          one_sec <= 0;
38          count_done <= '0';
39      elsif rising_edge(clk_2) then
40          if (f_state="010") then
41              if (stop_button='1') then
42                  count_done <= '1';
43              else
44                  one_millisecond := one_millisecond + 1;
45                  if one_millisecond = 9 then
46                      ten_millisecond := ten_millisecond + 1;
47                      one_millisecond := 0;
48                      if ten_millisecond = 9 then
49                          hundred_millisecond := hundred_millisecond + 1;
50                          ten_millisecond := 0;
51                          if hundred_millisecond = 9 then
52                              thousand_millisecond := thousand_millisecond + 1;
53                              hundred_millisecond := 0;
54                              count_done <= '1';
55                          end if;

```

## Random generator testbench



```
H:/Final/Avengers/Avengers.srscs/sources_1/new/Randomgen.vhd
1  -----
2
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7  entity Randomgen is
8  port (
9      rand:  out std_logic_vector(3 downto 0);
10     clock:  in  std_logic
11 );
12 end Randomgen;
13
14 architecture Behavioral of Randomgen is
15     signal state_now, state_next:  std_logic_vector(3 downto 0) := "1010";
16     signal feedback:               std_logic;
17     signal clock_line:             std_logic;
18 begin
19
20     process(clock_line, state_now, state_next)
21     begin
22         if rising_edge(clock_line) then
23             state_now <= state_next;
24         end if;
25     end process;
26     rand <= state_now;
27     clock_line <= clock;
28     feedback <= state_now(1) xor state_now(0);
29     state_next <= feedback & state_now(3 downto 1);
30 end Behavioral;
31
```

## Clock divider testbench

a.

```

9  entity Clockdivider is
10  port (
11      clock_1: in std_logic;
12      clock_2: out std_logic;
13      clock_3: out std_logic;
14      clock_4: out std_logic
15  );
16  end Clockdivider;
17
18  architecture Behavioral of Clockdivider is
19      signal line_2: std_logic := '0';
20      signal line_4: std_logic := '0';
21      signal line_3: std_logic := '0';
22  begin
23
24      clock_2 <= line_2;
25      clock_4 <= line_4;
26      clock_3 <= line_3;
27
28      FREQ_2: process(clock_1, line_2)
29          variable count_100000: unsigned (15 downto 0) := "1100001101010000";
30          variable counter_100000: unsigned (15 downto 0) := "0000000000000000";
31          begin
32              if (rising_edge(clock_1)) then
33                  if (counter_100000 = count_100000) then
34                      line_2 <= not line_2;
35                      counter_100000 := "0000000000000000";
36                  end if;
37                  counter_100000 := counter_100000 + 1;
38              end if;
39          end process;
40
41      FREQ_4: process(clock_1, line_4)
42          variable count_12500: unsigned (12 downto 0) := "1100001101010";
43          variable counter_12500: unsigned (12 downto 0) := "0000000000000";
44          begin
45              if (rising_edge(clock_1)) then
46                  if (counter_12500 = count_12500) then
47                      line_4 <= not line_4;
48                      counter_12500 := "0000000000000";
49                  end if;
50                  counter_12500 := counter_12500 + 1;
51              end if;
52          end process;
53
54      FREQ_3: process(clock_1, line_3)
55          variable count_100000000: unsigned (25 downto 0) := "101111101011110000100000000";

```



Design Verification:

Kindly find below the Video demonstration link

[Video demonstration](#)

## Conclusion

In summary, the project has successfully demonstrated the implementation of a timer circuit using VHDL programming and hardware design principles to measure the response time of the human hand to visual stimuli.

The functionality of the circuit is achieved through the integration of various modules, such as a clock divider, comparator, latch, counter and random generator, all controlled by a state machine.

The timer circuit follows specified requirements, allowing the user to initiate a test by pressing the start button, then a random amount of time elapses before the retry LED lights up and the counter runs in milliseconds.

The use of test benches for simulation ensures accurate verification of the functionality of each module and of the entire timing circuit.

Overall, the timer circuit has an efficient hardware design and VHDL programming skills that meet the project goals to accurately measure response times.

This project has potential applications in various fields, such as reaction time research and human-computer interface evaluation.

First, to improve the user experience, a display module can be added to show instructions or guide the user during testing.

Overall, the project has successfully demonstrated the integration of VHDL programming and hardware design knowledge to create an efficient timer circuit.

Through constant refinement and improvement, this timer circuit has the potential to find applications in various human performance evaluation scenarios and contribute to research and practical applications in various fields.

## appendix

---

-- Company: The Avengers

-- Engineer: Nomaan, Kingsley, Omar

---

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity final\_result is

Port(

start\_button: in std\_logic;

clear\_button: in std\_logic;

stop\_button: in std\_logic;

clock\_1: in std\_logic;

seven\_seg\_anode: out std\_logic\_vector(7 downto 0);

seven\_seg\_cathode: out std\_logic\_vector(7 downto 0);

led:out std\_logic\_vector(2 downto 0)

);

end final\_result;

architecture Behavioral of final\_result is

component capncomp is

```
    Port (  
        clock_3:    in std_logic;  
        random_number: in std_logic_vector(3 downto 0);  
        control:    in std_logic;  
        count_reached: out std_logic  
    );  
end component;
```

component Randomgen is

```
    port (  
        rand:  out std_logic_vector(3 downto 0);  
        clock: in std_logic  
    );  
end component;
```

component Clockdivider is

```
    port (  
        clock_1: in std_logic; --Clock 1 = 100MHz  
        clock_2: out std_logic; -- Clock 2 = 1000Hz(1KHz)  
        clock_3:  out std_logic;-- Clock 3 = 1Hz  
        clock_4: out std_logic -- Clock 4 = 8000Hz(8KHz)
```

```

);
end component;

component Sevensegmentd is
    Port (clk_4 : in std_logic;
          one_mils : in integer;
          ten_mils : in integer;
          hundred_mils : in integer;
          one_sec : in integer;
          f_state : in std_logic_vector(2 downto 0);
          anode : out std_logic_vector(7 downto 0);
          cathode : out std_logic_vector(7 downto 0)
    );
end component;

```

```

component Milsecouter is
    Port (clk_2 :in std_logic;
          clear_button : in std_logic;
          stop_button : in std_logic;
          f_state : in std_logic_vector(2 downto 0);
          one_mils : out integer;
          ten_mils : out integer;
          hundred_mils : out integer;
          one_sec : out integer;
    );
end component;

```

```

        count_done : out std_logic

    );

end component;


type state_type is (init, rand_wait, count, err, hold);
signal f_state : state_type := init;
signal f_state_parse: std_logic_vector(2 downto 0);


signal clock_3_line:    std_logic;
signal clock_2_line:    std_logic;
signal clock_4_line:    std_logic;
signal clock_1_line:    std_logic;


signal bus_random_number:  std_logic_vector(3 downto 0);


signal control_store_or_load:  std_logic;
signal control_count_reached:  std_logic;


signal reaction_time:  std_logic_vector(11 downto 0);


signal one_mils_bus, ten_mils_bus, hundred_mils_bus, one_sec_bus: integer;
signal count_done_bus : std_logic;

begin

```

```
clock_1_line <= clock_1;
```

```
Signalcount: capncomp port map(clock_3=>clock_3_line,  
random_number=>bus_random_number, control=>control_store_or_load,  
count_reached=>control_count_reached);
```

```
Random_gen: Randomgen port map(rand=>bus_random_number,  
clock=>clock_1_line);
```

```
Clock_divider: Clockdivider port map(clock_1=>clock_1_line, clock_2=>clock_2_line,  
clock_3=>clock_3_line, clock_4=>clock_4_line);
```

```
Seven_segmentd: Sevensegmentd port map (clk_4=>clock_4_line,  
one_mils=>one_mils_bus, ten_mils=>ten_mils_bus,  
hundred_mils=>hundred_mils_bus, one_sec=>one_sec_bus,  
f_state=>f_state_parse, anode=>seven_seg_anode,  
cathode=>seven_seg_cathode);
```

```
Mil_secouter: Milsecouter port map (clk_2 => clock_2_line, clear_button =>  
clear_button, stop_button => stop_button, f_state => f_state_parse, one_mils =>  
one_mils_bus, ten_mils => ten_mils_bus, hundred_mils => hundred_mils_bus,  
one_sec => one_sec_bus, count_done => count_done_bus);
```

```
STATE_HANDLER:  process (clock_1_line, clear_button, stop_button, start_button)
```

```
begin
```

```
    if (clear_button='1') then
```

```
        f_state <= init;
```

```
        led <= "000";
```

```

elsif rising_edge(clock_1_line) then
  case f_state is
    when init =>
      control_store_or_load <= '1';
      f_state_parse <= "000";
      if (start_button='1') then
        f_state<=rand_wait;

      end if;

    when rand_wait =>
      f_state_parse <= "001";
      control_store_or_load <= '0';
      if (stop_button='1') then
        f_state <= err;
      elsif (clear_button='1') then
        f_state <= init;
      elsif (control_count_reached='1') then
        f_state <= count;
      end if;
    when count =>

      led <= "111";

      f_state_parse <= "010";

```



```

control_store_or_load <= '1';

if (stop_button='1') then

    f_state<=hold;

elsif (clear_button='1') then

    f_state<=init;


elsif (count_done_bus = '1') then

    f_state<=hold;

end if;


when err =>

    f_state_parse <= "011";

    control_store_or_load <= '1';

    if (clear_button='1') then

        f_state<=init;

    end if;

when hold =>

    f_state_parse <= "100";

    led <= "000";

    control_store_or_load <= '1';

    if (clear_button='1') then

        f_state<=init;

    end if;

when others =>

```

```

        f_state <= init;

        control_store_or_load <= '1';

    end case;

end if;

end process;

end Behavioral;

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;

```

```

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

```

```

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

```

```

entity capncomp is

```

```

    Port (

```

```

    clock_3:    in std_logic;

    random_number: in std_logic_vector(3 downto 0);

    control:    in std_logic;

    count_reached: out std_logic

);

end capncomp;

```

architecture Behavioral of capncomp is

component four\_bit\_latch is

```

    port (

        D: in std_logic_vector(3 downto 0);

        E: in std_logic;

        Q: out std_logic_vector(3 downto 0)

    );

end component;

```

component comparator\_fourbit is

```

    port (

        a:    in std_logic_vector(3 downto 0);

        b:    in std_logic_vector(3 downto 0);

        gtoe: out std_logic;

        clock: in std_logic;

```

```

        pause: in std_logic

    );

end component;

component counter_wrapper is

    port (

        CLK : in STD_LOGIC;

        Q : out STD_LOGIC_VECTOR ( 3 downto 0 );

        SCLR : in STD_LOGIC

    );

end component;

signal latch_to_comparator_input_b:   std_logic_vector(3 downto 0);

signal count_to_comparator_input_a:   std_logic_vector(3 downto 0);

begin

CCOMPARE: four_bit_latch port map(D=>random_number, E=>control,
Q=>latch_to_comparator_input_b);

CCOUNTER: counter_wrapper port map(CLK=>clock_3,
Q=>count_to_comparator_input_a, SCLR=>control);

COMPARAT: comparator_fourbit port map(a=>count_to_comparator_input_a,
b=>latch_to_comparator_input_b, gtoe=>count_reached, clock=>clock_3,
pause=>control);

end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity Randomgen is
port (
    rand:  out std_logic_vector(3 downto 0);
    clock: in std_logic
);
end Randomgen;
```

```
architecture Behavioral of Randomgen is
    signal state_now, state_next: std_logic_vector(3 downto 0) := "1010";
    signal feedback:             std_logic;
    signal clock_line:           std_logic;
begin

    process(clock_line, state_now, state_next)
```

```

begin
    if rising_edge(clock_line) then
        state_now <= state_next;
    end if;
end process;

rand <= state_now;

clock_line <= clock;

feedback <= state_now(1) xor state_now(0);

state_next <= feedback & state_now(3 downto 1);

end Behavioral;

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;

```

```

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

```

```

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.

--library UNISIM;

```

```
--use UNISIM.VComponents.all;
```

```
entity Clockdivider is
```

```
port (
```

```
    clock_1: in std_logic;
```

```
    clock_2: out std_logic;
```

```
    clock_3:  out std_logic;
```

```
    clock_4: out std_logic
```

```
);
```

```
end Clockdivider;
```

```
architecture Behavioral of Clockdivider is
```

```
    signal line_2:  std_logic := '0';
```

```
    signal line_4:  std_logic := '0';
```

```
    signal line_3:  std_logic := '0';
```

```
begin
```

```
    clock_2 <= line_2;
```

```
    clock_4 <= line_4;
```

```
    clock_3 <= line_3;
```

```
FREQ_2:  process(clock_1, line_2)
```

```
    variable count_100000:  unsigned (15 downto 0) := "1100001101010000";
```

```
    variable counter_100000:  unsigned (15 downto 0) := "0000000000000000";
```

```

begin
    if (rising_edge(clock_1)) then
        if (counter_100000 = count_100000) then
            line_2 <= not line_2;
            counter_100000 := "000000000000000000";
        end if;
        counter_100000 := counter_100000 + 1;
    end if;
end process;

```

```

FREQ_4:  process(clock_1, line_4)
    variable count_12500:  unsigned (12 downto 0) := "1100001101010";
    variable counter_12500:  unsigned (12 downto 0) := "00000000000000";
begin
    if (rising_edge(clock_1)) then
        if (counter_12500 = count_12500) then
            line_4 <= not line_4;
            counter_12500 := "00000000000000";
        end if;
        counter_12500 := counter_12500 + 1;
    end if;
end process;

```

```

FREQ_3:  process(clock_1, line_3)

```



```

        variable count_100000000: unsigned (25 downto 0) :=
"10111110101111000010000000";

        variable counter_100000000: unsigned (25 downto 0) :=
"0000000000000000000000000000";

begin

    if (rising_edge(clock_1)) then

        if (counter_100000000 = count_100000000) then

            line_3 <= not line_3;

            counter_100000000 := "0000000000000000000000000000";

        end if;

        counter_100000000 := counter_100000000 + 1;

    end if;

end process;

end Behavioral;

```

---

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```

```
entity four_bit_latch is  
  
    port (  
  
        D: in std_logic_vector(3 downto 0);  
  
        E: in std_logic;  
  
        Q: out std_logic_vector(3 downto 0)  
  
    );  
end four_bit_latch;
```

architecture Behavioral of four\_bit\_latch is

```
begin  
  
    process (D, E)  
  
        begin  
  
            if (E='1') then  
  
                if (D = "0001") then  
  
                    Q <= "1000";  
  
                else  
  
                    Q(0)<=D(0);  
  
                    Q(1)<=D(1);  
  
                    Q(2)<=D(2);  
  
                    Q(3)<=D(3);  
  
                end if;  
  
            end if;  
  
        end process;
```

```
end Behavioral;
```

-----

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity counter_wrapper is
```

```
port (
```

```
    CLK: in STD_LOGIC;
```

```
    Q: out STD_LOGIC_VECTOR ( 3 downto 0 );
```

```
    SCLR: in STD_LOGIC
```

```
);
```

```
end counter_wrapper;
```

```
architecture Behavioral of counter_wrapper is
```

```
component counter is
```

```
port (
```

```
    Q: out STD_LOGIC_VECTOR ( 3 downto 0 );
```

```
    SCLR: in STD_LOGIC;
```

```
    CLK: in STD_LOGIC
```

```
);
```

```

        end component counter;

begin

counter_i: component counter port map (CLK => CLK, Q(3 downto 0) => Q(3 downto
0), SCLR => SCLR);

end Behavioral;

```

-----

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;

```

```

entity counter is

port (

    Q:    out STD_LOGIC_VECTOR ( 3 downto 0 );

    SCLR: in STD_LOGIC;

    CLK:  in STD_LOGIC

);

end counter;

```

```

architecture Behavioral of counter is

signal count: std_logic_vector(0 to 3);

begin

```

```

process(SCLR,clk)

begin

    if (SCLR = '1') then count <= "0000";

    elsif (clk'event and clk = '1') then count <= count + 1;

    end if;

end process;

Q <= count;

end Behavioral;

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```

```

entity comparator_fourbit is

port (

    a:    in std_logic_vector(3 downto 0);

    b:    in std_logic_vector(3 downto 0);

    gtoe: out std_logic;

    clock: in std_logic;

```

```

        pause: in std_logic
    );
end comparator_fourbit;

architecture Behavioral of comparator_fourbit is

begin

    process(clock, pause, a, b)
    begin
        if (pause='1') then
            gtoe <= '0';
        elsif (a >= b) then
            gtoe <= '1';
        else
            gtoe <= '0';
        end if;
    end process;
end Behavioral;

```

---

```

library IEEE;

```

```
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity Randomgen is
```

```
port (
```

```
    rand: out std_logic_vector(3 downto 0);
```

```
    clock: in std_logic
```

```
);
```

```
end Randomgen;
```

```
architecture Behavioral of Randomgen is
```

```
    signal state_now, state_next: std_logic_vector(3 downto 0) := "1010";
```

```
    signal feedback:          std_logic;
```

```
    signal clock_line:        std_logic;
```

```
begin
```

```
    process(clock_line, state_now, state_next)
```

```
    begin
```

```
        if rising_edge(clock_line) then
```

```
            state_now <= state_next;
```

```
        end if;
```

```
    end process;
```

```
    rand <= state_now;
```

```
    clock_line <= clock;
```

```
    feedback <= state_now(1) xor state_now(0);
```

```
    state_next <= feedback & state_now(3 downto 1);  
end Behavioral;
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.all;  
use IEEE.STD_LOGIC_UNSIGNED.all;
```

```
entity Clockdivider is
```

```
    port (  
        clock_1: in std_logic;  
        clock_2: out std_logic;  
        clock_3:  out std_logic;  
        clock_4: out std_logic  
    );
```

```
end Clockdivider;
```

```
architecture Behavioral of Clockdivider is
```

```
    signal line_2: std_logic := '0';  
    signal line_4: std_logic := '0';  
    signal line_3: std_logic := '0';
```



```
begin
```

```
clock_2 <= line_2;
```

```
clock_4 <= line_4;
```

```
clock_3 <= line_3;
```

```
FREQ_2: process(clock_1, line_2)
```

```
variable count_100000: unsigned (15 downto 0) := "1100001101010000";
```

```
variable counter_100000: unsigned (15 downto 0) := "0000000000000000";
```

```
begin
```

```
if (rising_edge(clock_1)) then
```

```
if (counter_100000 = count_100000) then
```

```
line_2 <= not line_2;
```

```
counter_100000 := "0000000000000000";
```

```
end if;
```

```
counter_100000 := counter_100000 + 1;
```

```
end if;
```

```
end process;
```

```
FREQ_4: process(clock_1, line_4)
```

```
variable count_12500: unsigned (12 downto 0) := "1100001101010";
```

```
variable counter_12500: unsigned (12 downto 0) := "00000000000000";
```

```
begin
```

```
if (rising_edge(clock_1)) then
```

```

        if (counter_12500 = count_12500) then

            line_4 <= not line_4;

            counter_12500 := "00000000000000";

        end if;

        counter_12500 := counter_12500 + 1;

    end if;

end process;

```

```

FREQ_3:    process(clock_1, line_3)

    variable count_100000000: unsigned (25 downto 0) :=
"10111110101111000010000000";

    variable counter_100000000: unsigned (25 downto 0) :=
"0000000000000000000000000000";

    begin

        if (rising_edge(clock_1)) then

            if (counter_100000000 = count_100000000) then

                line_3 <= not line_3;

                counter_100000000 := "0000000000000000000000000000";

            end if;

            counter_100000000 := counter_100000000 + 1;

        end if;

    end process;

end Behavioral;

```

```

-----

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;


entity Sevenssegmentd is
    Port (clk_4 : in std_logic;
          one_mils : in integer;
          ten_mils : in integer;
          hundred_mils : in integer;
          one_sec : in integer;
          f_state : in std_logic_vector(2 downto 0);
          anode : out std_logic_vector(7 downto 0);
          cathode : out std_logic_vector(7 downto 0)
    );
end Sevenssegmentd;


architecture Behavioral of Sevenssegmentd is
begin
    process(clk_4)

```

```

    variable seg_sel: unsigned (1 downto 0) := "00";

begin

    if rising_edge(clk_4) then

        if f_state = "000" then

            case (seg_sel) is

                when "00" =>

                    anode <= "11110111";

                    cathode <= "11111111";

                when "01" =>

                    anode <= "11111101";

                    cathode <= "10010001";

                when "10" =>

                    anode <= "11111110";

                    cathode <= "11110011";

                when "11" =>

                    anode <= "11111110";

                    cathode <= "11111111";

                when others =>

                    anode <= "11110000";

                    cathode <= "11111110";

            end case;

            seg_sel := seg_sel + 1;

        elsif f_state = "001" then

            anode <= "11110000";

```

```

        cathode <= "11111111";
    elsif ((f_state = "010") or (f_state = "100")) then
        case (seg_sel) is
            when "00" =>
                anode <= "11110111";
                case (one_sec) is
                    when 0 =>
                        cathode <= "00000011";
                    when 1 =>
                        cathode <= "10011111";
                    when others =>
                        cathode <= "11111110";
                end case;
            when "01" =>
                anode <= "11111011";
                case (hundred_mils) is
                    when 0 =>
                        cathode <= "00000011";
                    when 1 =>
                        cathode <= "10011111";
                    when 2 =>
                        cathode <= "00100101";
                    when 3 =>
                        cathode <= "00001101";

```

```

when 4 =>
    cathode <= "10011001";
when 5 =>
    cathode <= "01001001";
when 6 =>
    cathode <= "11000001";
when 7 =>
    cathode <= "00011111";
when 8 =>
    cathode <= "00000001";
when 9 =>
    cathode <= "00011001";
when others =>
    cathode <= "11111110";
end case;

when "10" =>
    anode <= "11111101";

    case (ten_mils) is
        when 0 =>
            cathode <= "00000011";
        when 1 =>
            cathode <= "10011111";
        when 2 =>
            cathode <= "00100101";
    end case;

```

```

when 3 =>
    cathode <= "00001101";
when 4 =>
    cathode <= "10011001";
when 5 =>
    cathode <= "01001001";
when 6 =>
    cathode <= "11000001";
when 7 =>
    cathode <= "00011111";
when 8 =>
    cathode <= "00000001";
when 9 =>
    cathode <= "00011001";
when others =>
    cathode <= "11111110";
end case;

when "11" =>
    anode <= "11111110";
case (one_mils) is
when 0 =>
    cathode <= "00000011";
when 1 =>
    cathode <= "10011111";

```

```

when 2 =>
    cathode <= "00100101";
when 3 =>
    cathode <= "00001101";
when 4 =>
    cathode <= "10011001";
when 5 =>
    cathode <= "01001001";
when 6 =>
    cathode <= "11000001";
when 7 =>
    cathode <= "00011111";
when 8 =>
    cathode <= "00000001";
when 9 =>
    cathode <= "00011001";
when others =>
    cathode <= "11111110";
end case;

when others =>
    anode <= "11110000";
    cathode <= "11111110";
end case;

seg_sel := seg_sel + 1;

```



```

        elsif f_state = "011" then
            anode <= "11110000";
            cathode <= "00011001";

        end if;

    end if;

end process;

end Behavioral;

```

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

```

entity Milsecounter is

```

Port (clk_2 :in std_logic;

      clear_button : in std_logic;

      stop_button : in std_logic;

      f_state : in std_logic_vector(2 downto 0);

      one_mils : out integer;

      ten_mils : out integer;

      hundred_mils : out integer;

```

```

        one_sec : out integer;

        count_done : out std_logic

    );

end Milsecouter;

```

architecture Behavioral of Milsecouter is

begin

```
Milsecouter: process (clk_2, clear_button, stop_button, f_state)
```

```
variable one_millisecond : integer := 0;
```

```
variable ten_millisecond : integer := 0;
```

```
variable hundred_millisecond : integer := 0;
```

```
variable thousand_millisecond : integer := 0;
```

```
begin
```

```
    if (clear_button='1') then
```

```
        one_millisecond := 0;
```

```
        ten_millisecond := 0;
```

```
        hundred_millisecond := 0;
```

```
        thousand_millisecond := 0;
```

```
        one_mils <= 0;
```

```
        ten_mils <= 0;
```

```
        hundred_mils <= 0;
```

```
        one_sec <= 0;
```

```
        count_done <= '0';
```

```
    elsif rising_edge(clk_2) then
```

```

if (f_state= "010") then
    if (stop_button='1') then
        count_done <= '1';
    else
        one_millisecond := one_millisecond + 1;
        if one_millisecond = 9 then
            ten_millisecond := ten_millisecond + 1;
            one_millisecond := 0;
            if ten_millisecond = 9 then
                hundred_millisecond := hundred_millisecond + 1;
                ten_millisecond := 0;
                if hundred_millisecond = 9 then
                    thousand_millisecond := thousand_millisecond + 1;
                    hundred_millisecond := 0;
                    count_done <= '1';
                end if;
            end if;
        end if;
    end if;
end if;

one_mils <= one_millisecond;
ten_mils <= ten_millisecond;
hundred_mils <= hundred_millisecond;

```

```
        one_sec <= thousand_millisecond;  
    end process;  
  
end Behavioral;
```